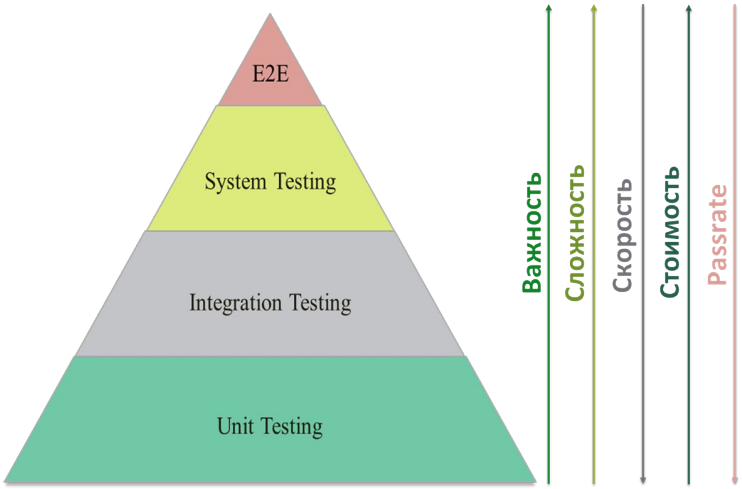
«Test_Pyramid»

Пирамида тестирования, также часто говорят уровни тестирования, это группировка тестов по уровню детализации и их назначению.

Пирамиду разбивают на 4 уровня (снизу вверх):

- модульное тестирование (юнит);
- интеграционное тестирование;
- системное тестирования;
- приемочное тестирование.



Но можно встретить варианты, где 3 уровня В этой модели объединяют интеграционный и системный уровни.

Можно сказать, что разработка ПО - это движение по пирамиде снизу вверх. Важно отметить:

1. Тест *(ручной, на высоких уровнях, или автотест, на низких уровнях)*, должен быть на том же уровне, что и тестируемый объект. Например, модульный тест *(проверяющий функции, классы, объекты и т.п.)* должен быть на компонентном уровне. Это неправильно, если на приемочном уровне запускается тест, который проверят минимальную единицу кода.
2. Тесты уровнем выше не проверяют логику тестов уровнем/уровнями ниже.
3. Чем выше тесты уровнем, тем они:

- сложней в реализации, и соответственно, дороже в реализации;
- важнее для бизнеса и критичней для пользователей;

- замедляют скорость прохождения тестовых наборов, например, регресса.

Модульное тестирование

Чаще всего называют юнит тестированием. Реже называют модульным тестированием. На этом уровне тестируют атомарные части кода. Это могут быть классы, функции или методы классов.
Пример: твоя компания разрабатывает приложение "Калькулятор", которое умеет складывать и вычитать. Каждая операция это одна функция. Проверка каждой функции, которая не зависит от других, является юнит тестированием.

Юнит тесты находят ошибки на фундаментальных уровнях, их легче разрабатывать и поддерживать. Важное преимущество модульных тестов в том, что они быстрые и при изменении кода позволяют быстро провести регресс *(убедиться, что новый код не сломал старые части кода).*

Тест на этом уровне:

1. Всегда автоматизируют.
2. Модульных тестов всегда больше, чем тестов с других уровней.
3. Юнит тесты выполняются быстрее всех и требуют меньше ресурсов.
4. Практически всегда юнит тесты не зависят от других модулей *(на то они и юнит тесты)* и UI системы.

В 99% разработкой модульных тестов занимается разработчик, при нахождении ошибки на этом уровне не создается баг-репортов. Разработчик находит баг, правит, запускает и проверяет *(абстрактно говоря это разработка через тестирование)* и так по новой, пока тест не будет пройден успешно.

На модульном уровне разработчик (или автотестер) использует метод белого ящика. Он знает что принимает и отдает минимальная единица кода, и как она работает.

Существуют десятки фреймворков для модульного тестирования, доступных для различных языков программирования. Некоторые популярные среды модульного тестирования включают Cunit, Moq, Cucumber, Selenium,

Embunit, Sass True, HtmlUnit и JUnit (одна из наиболее широко используемых сред с открытым исходным кодом для языка программирования Java).

Интеграционный уровень

Определяется как тип тестирования, при котором программные модули интегрируются логически и тестируются как группа.

Проверяет взаимосвязь компонентов, которые проверяли на модульном уровне, с другой или другими компонентами, а также интеграцию компонентов с системой *(проверка работы с ОС, сервисами и службами, базами данных, железом и т.д.)*.

Это проверки API, работы сервисов *(проверка логов на сервере, записи в БД)* и т.п.

На этом уровне используется либо серый, либо черный ящик.

В интеграционном тестировании есть 3 основных способа тестировани*)*:

- Снизу вверх (Bottom Up Integration): все мелкие части модуля собираются в один модуль и тестируются. Далее собираются следующие мелкие модули в один большой и тестируется с предыдущим и т.д. *Например, функция публикации фото в соц. профиле состоит из 2 модулей: загрузчик и публикатор. Загрузчик, в свою очередь, состоит из модуля компрессии и отправки на сервер. Публикатор состоит из верификатора (проверяет подлинность) и управления доступом к фотографии. В интеграционном тестировании соберем модули загрузчика и проверим, потом соберем модули публикатора, проверим и протестируем взаимодействие загрузчика и публикатор.*
- Сверху вниз (Top Down Integration): сначала проверяем работу крупных модулей, спускаясь ниже добавляем модули уровнем ниже. На этапе проверки уровней выше данные, необходимые от уровней ниже, симулируются.*Например, проверяем работу загрузчика и публикатора. Руками (создаем функцию-заглушку) передаем от загрузчика публикатору фото, которое якобы было обработано компрессором.*
- Большой взрыв ("Big Bang" Integration): собираем все реализованные модули всех уровней, интегрируем в систему и тестируем. Если что-то не работает или недоработали, то фиксим или дорабатываем.

Системный уровень

Системное тестирование используется для тестирования полной сборки продукта со всеми компонентами вместе. В то время как интеграционное тестирование тестирует модули связанных компонентов, системное тестирование проверяет, как программа работает со всеми интегрированными модулями, и выявляет дефекты в межмодульных операциях.

Например, мы сначала интегрируем все наши программные модули, такие как вход в учетную запись, поисковый веб-сайт и т. Д., А затем подключим все модули и запускаем тестовые примеры через программу, например «создать учетную запись и опубликовать сообщение на форуме». Системное тестирование часто выполняется отдельной командой тестирования, чтобы избежать предвзятости подтверждения разработчика.

Системное тестирование проводится следующим образом.

1. Сперва создается тестовый план, затем тест- и юз-кейсы.
2. Следующим шагом создаются тестовые данные, используемые для системного тестирования, и проводится автоматизированный прогон тест-кейсов, затем обычный.
3. По результатам тестирования формируется отчет по выявленным ошибкам и проводится регрессионное тестирование после их исправления.

Цикл повторяется до исключения всех ошибок.

Системное тестирование также включает тесты:

- на соответствие функциональным и нефункциональным требованиям,
- адаптационное тестирование,
- нагрузочное и стрессовое тестирование,
- юзабилити тестирование,
- тестирование совместимости,
- тестирование пользовательского интерфейса,
- тестирование производительности, и др.

По завершению системного тестирования мы предоставляем подробный отчет о найденных ошибках и рекомендациях по их устранению.

Приемочное тестирование

Также часто называют E2E тестами (End-2-End) или сквозными. На этом уровне происходит валидация требований *(проверка работы ПО в целом, не только по прописанным требованиям, что проверили на системном уровне).*

Приемочное тестирование (или пользовательское приемочное тестирование) — это тест, выполняемый на поздних этапах процесса разработки, чтобы оценить, все ли первоначально заданные требования удовлетворены окончательной сборкой продукта.

Проверка требований производится на наборе приемочных тестов. Они разрабатываются на основе требований и возможных способах использования ПО.

Важно помнить, что E2E тесты автоматизируются сложнее, дольше, стоят дороже, сложнее поддерживаются и трудно выполняются при регрессе. Значит таких тестов должно быть меньше.

Как внутренние, так и внешние тестировщики проверяют исходные спецификации продукта и бизнес-требования, а затем отмечают каждый из них по мере использования продукта. Есть много способов проведения приемочного тестирования, наиболее распространенными из которых являются альфа-тестирование (внутреннее) и бета-тестирование (внешнее).